VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014

Martin Babinec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Babinec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ITA spol. s r.o.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Peter Chovanec**

Konzultant bakalářské práce: Ing. Pavel Šimeček, Ph.D.

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: 6.5.2014

Marian Babinec
.....
podpis studenta

Rád bych na tomto místě poděkoval mému konzultantu panu Ing. Pavlu Šimečkovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce, programátorům Bc. Jakubu Klebercovi a Bc. Jiřímu Stařinskému za pomoc při řešení spousty problémů, které se vyskytly v průběhu a mému vedoucímu bakalářské práce Ing. Peterovi Chovancovi.

Abstrakt

Práce pojednává o absolvování odborné praxe ve firmě ITA spol. s.r.o. Cílem je popsat práci na zadaných úkolech a jejich řešení. Bakalářská práce také pojednává o získaných zkušenostech z průběhu vykonávání praxe a vymezuje uplatněné znalosti ze studia.

Klíčová slova: teplotní simulace, metalurgie, C++, C#

Abstract

This thesis describes duration of professional practice in the company ITA spol. s.r.o. The aim is description of work on assigned tasks and their solutions. Bachelor thesis is also about gained experiences from duration of practice and specify used knowledge from study.

Keywords: simulation of temperature, metallurgy, C++, C#

Seznam použitých zkratek a symbolů

OpenGL	–	Open Graphics Library
GUI	–	Graphical User Interface
MKP	–	Metoda Konečných Prvků
UML	–	Unified Modeling Language
RP	–	Reference Point
CPU	–	Central Processing Unit
P/Invoke	–	Platform Invocation Services

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V této práci je mým cílem seznámit čtenáře s průběhem vykonávání odborné praxe v ostravské firmě ITA spol. s r.o. Velice jsem usiloval o vykonání praxe v nějaké firmě místo klasického psaní bakalářské práce protože jsem si uvědomoval nutnost reálně nabytých zkušeností pro budoucí snažší uplatnění na pracovním trhu. Jelikož počet zájemců o praxi byl větší než počet míst, bylo nám všem zadáno napsání programu v OpenGL který vykresloval trojrozměrnou plochu podle zadaných hodnot v excelovém souboru. Podle výsledné kvality tohoto programu došlo k finálnímu náboru dvou zájemců o praxi z pěti, mezi kterými jsem byl i já.

2 Popis odborného zařazení firmy a pracovního zařazení studenta

Společnost ITA spol. s r.o. byla založena v roce 1991 výzkumnými a vědeckými pracovníky Výzkumného ústavu strojírenského a metalurgického Vítkovice a současnými vlastníky společnosti panem Ing. Pavlem Šimečkem, PhD. a panem Ing. Danielem Hajdukem, PhD. [1]

Společnost se zabývá oblastmi:

- moderními technologiemi válcování za tepla i za studena
- dodává know-how a programová řešení významným dodavatelům válcovacích zařízení, technologií a řídicích systémů jako např. Danieli, ConverTeam, Acos Vilares, Vítkovice Heavy Machinery..
- řeší technické problémy a technologické inovace na teplých a studených válcovacích tratích jako např. ArcelorMittal Ostrava, Severstal Čerepovec, ArcelorMittal Vanderbijlpark, CSN Voltaredunda, Třinecké železárny Třinec..

2.1 Pracovní zařazení studenta

Ve firmě jsem nastoupil do kolektivu na pozici C++ a částečně taky C# programátora. Ve firmě pracují celkově dva programátoři, kteří mají hlavně na starost vývoj a úpravu GUI pro matematické funkce napsané p.Šimečkem a p.Hajdukem. Z počátku jsem znal pouze první ze dvou prací, které jsem vykonal. Na seznamu byly potenciální problémy, na kterých bych mohl pracovat po dokončení první práce, nakonec má druhá práce ze seznamu vůbec nepocházela.

Jelikož všechny komerční programy firmy jsou napsáné v C++, také většina mého kódu byla napsána v C++, protože se počítalo s tím, že mé funkce budou skutečně nasazeny v těchto programech. V určitých případech jsem však měl volnou ruku a pokud jsem mohl zvolil jsem programovací jazyk C#.

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe, jejich postupy a vyjádření časové náročnosti

3.1 Funkce pro výpočet teplot v průřezu tělesa

Má první práce ve firmě spočívala ve vývoji algoritmu funkce pro přibližnou aproximaci nerovnoměrného teplotního pole z diskrétních hodnot v různých hloubkách pod povrchem do uzlů MKP sítě tepelně zpracovávaného tělesa.

3.1.1 Problém

Program QTSteel, určený pro výpočtové modelování procesů tepelného zpracování ocelí, počítá teplotní křivky při ochlazování tepelně zpracovávaných těles pomocí 2D MKP [2] (příklad sítě MKP prvků tělesa najdete v příloze 12). Pro tyto nestacionární teplotní výpočty je nutné znát rozložení počáteční teploty v uzlech MKP sítě daného tělesa. Stávající verze programu umožňovala pouze předepsání konstantní počáteční teploty. Nová funkce, vytvořená v rámci bakalářské praxe, umožnila předepsat nerovnoměrné rozložení počáteční teploty na základě znalosti přibližné polohy teplotních izočar (například z měření teploty pomocí termovize).

3.1.2 Zadání

Máme průřez 2D tělesem pokrytého sítí bodů (tzv. MKP síť) a konstantní tabulku teplot v předepsaných hloubkách pod povrchem tohoto tělesa (tzv. isočáry). Je potřeba napsat funkci, která vypočítá teploty všech uzlů sítě tak, aby teplota tělesa v dané hloubce pod obrysem tělesa co nejlépe odpovídala předepsané teplotní tabulce.

3.1.3 Vstupní data

- Teplotní tabulka - hloubka v tělesu - teplota v hloubce
- Souřadnice X,Y bodů v průřezu tělesa
- Tabulka indexů bodů vnějších a vnitřních hran obrysu tělesa

3.1.4 Požadovaný výstup

Požadovaným výstupem funkce bylo pole vypočítaných teplot v uzlech podle zadané teplotní tabulky a zadaných souřadnic uzlů.

3.1.5 Řešení

Po obdržení zadání jsem začal přemýšlet nad možným principem algoritmu. Po analýze problému jsem zjistil, že problém je relativně jednoduchý - najít ke každému vnitřnímu bodu nejbližší vnější hranu tělesa, vypočítat vzdálenost mezi bodem a hranou a ze

vzdálenosti poté interpolovat teploty podle tabulky teplota-hloubka. Po několika menších změnách, kdy algoritmus neřešil všechny body, jsem došel k finálnímu algoritmu, který byl neformálně definován následujícím způsobem.

Definice pojmů

- Obrys - zadaná uzavřená polyčára, Obrys se skládá z hran Obrysu
- Uzel - zadaný bod ležící uvnitř Obrysu
- ISO-hodnota - zadaná dvojce - vzdálenost od hrany Obrysu, odpovídající teplota
- ISO-vzdálenost - stanovená vzdálenost Uzlu od Obrysu
- T-hodnota - stanovená teplota v Uzlu
- KK bod - bod Obrysu, který tvoří vrchol úhlu hran Obrysu $> \pi$
- Kolmá oblast - oblast vymezená hranou Obrysu a polopřímkami vycházejícími kolmo z bodů hrany ve směru dovnitř Obrysu

Cíl

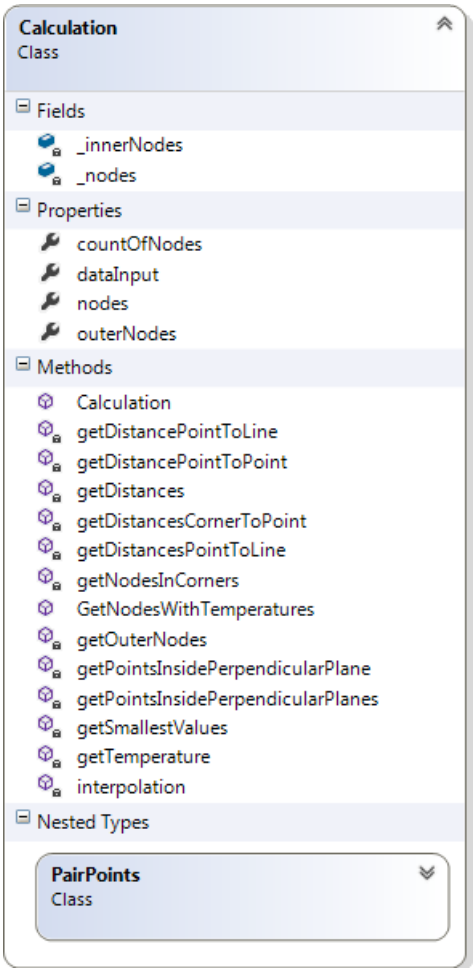
Stanovit T-hodnotu v Uzlu na základě jeho polohy uvnitř Obrysu a souboru ISO-hodnot.

Algoritmus

1. Každému Uzlu najít minimální vzdálenost od hrany Obrysu takovou, že Uzel leží v její Kolmé oblasti
2. Každému Uzlu spočítat jeho vzdálenost od nejbližšího KK bodu.
3. Každému Uzlu stanovit ISO-vzdálenost jako MINIMUM{vzdálenost z Kolmé oblasti, vzdálenosti od hran KK bodu}
4. Převést ISO-vzdálenost na T-hodnotu na základě předepsaného souboru ISO-hodnot.

Touto neformální definicí jsem si ujasnil jak algoritmus má pracovat. Z důvodu rychlejšího vývoje algoritmu jsem zvolil postup naprogramování kódu v mnou oblíbeném jazyku C#, ve kterém jsem se nemusel nijak zabývat syntaxí jazyku C++, a následného přepsání kódu do jazyku C++.

Kód je napsaný zcela objektovým způsobem. Všechny výpočty jsou zapouzdřené v jedné hlavní třídě Calculate, jejiž UML diagram můžete vidět na obrázku 1.



Obrázek 1: UML diagram třídy Calculate

V následujícím výpisu kódu 1 uvádím příklad načtení dat ze souborů a získání vypočítaných teplot v bodech sítě.

```
List<CNode> nodes = DataLoader.LoadPoints("points.txt");
CDataInput dataInput = DataLoader.LoadEdges("edges.txt", nodes);
var isoTable = DataLoader.LoadIso("iso.txt");

Calculation calc = new Calculation(dataInput);
List<CNode> nodesWithTemparatures = calc.GetNodesWithTemperatures(iso);
```

Výpis 1: Nahrání vstupních dat a získání výsledku z třídy Calculate

Jak můžete vidět, získání výpočtů je po předání vstupních dat maximálně jednoduché.

Důležitou třídou je také statická třída DataLoader, která má metody pro načtení dat z textových souborů, jejíž formát jsem dostal předem zadaný. Tato třída není v programu QTSteel potřeba, data se předávají v již potřebných struktůrách.

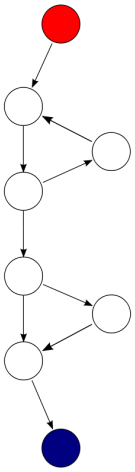
Příklad průřezu tělesa vykreslené barevným spektrem podle vypočítaných teplot v bodech MKP sítě z mé funkce najdete v příloze 11.

3.2 Cyklomatická komplexita

Napsaný kód jsem se snažil udržet v co největší jednoduchosti a čistotě kódu. Pro tyto účely bylo zavedeno několik metrik, které programátorům napovídají jak na tom jejich kód je a zda nepotřebuje být refaktorován.

Jednou z metrik kvality a čistoty kódu je měření tzv. cyklomatické komplexity (také cyklomatická složitost). Toto číslo vyjadřuje složitost programu a to konkrétně tak, že měří počet možných cest skrz zdrojový kód.

Nižší číslo znamená nižší možnost zanesení chyb do kódu v případě úprav, snadnější pochopení kódu a také snadnější testování. Na následujícím příkladu se pokusím vysvětlit počítání cyklomatické komplexity.



Obrázek 2: Příklad průběhu programu. Obrázek převzat z internetu [3].

Na obrázku 2 je zobrazený grafem jednoduchý příklad toku programu. Graf začíná červeným uzlem, kde hned poté vstoupí do cyklu (vyjádřený třemi uzly hned pod ním). Aby se z cyklu dalo vystoupit, musí být splněna (resp. nesplněna) podmínka, cyklomatické číslo se zvýší o 1. Po ukončení cyklu se přejde k podmínce, kde tok programu může pokračovat dvěmi cestami, cyklomatická složitost se tedy zvýší o 2. Program dojde v modrém bodu do svého konce. Celková cyklomatická složitost u tohoto programu je 3.

Měření různých metrik nám velice zjednodušují moderní vývojová prostředí. Samotné Microsoft Visual Studio, ve kterém jsem program psal, umí spočítat nejen cyklomatickou složitost ale ještě další čtyři metriky a to - index udržitelnosti, hloubku dědění, počet řádků kódu a tzv. class coupling. Ke všem těmto metrikám se dostaneme ve Visual Studio 2008 a novější kliknutím pravým tlačítkem v oknu Solution Exploreru na projekt a vybrání možnosti Calculate Code Metrics. Po krátké chvíli (záleží na velikosti projektu) se zobrazí okno s naměřenými hodnotami jako na obrázku 5.

Code Metrics Results

Filter: None Min: 0 Max: <Maximum Value>

Hierarchy	Maintainability Index	Cyclomatic Complexity
Calculation	66	62
nodes.get() : List<CNode>	98	1
outerNodes.get() : List<CNode>	98	1
dataInput.get() : CDataInput	98	1
countOfNodes.get() : int	98	1
nodes.set(List<CNode>) : void	95	1
outerNodes.set(List<CNode>) : void	95	1
dataInput.set(CDataInput) : void	95	1
countOfNodes.set(int) : void	95	1
interpolation(double, double, double, double, double) : double	82	1
getDistancePointToPoint(double, double, double, double) : doubl	81	1
getDistances() : double[]	72	1
GetNodesWithTemperatures(List<KeyValuePair<double, double>	70	2
getDistancePointToLine(Point, Line) : double	67	1
getTemperature(double, List<KeyValuePair<double, double>>) :	67	4
Calculation(CDataInput)	67	1
getSmallestValues(double[], double[]) : double[]	63	5
getNodesInCorners() : int[]	60	5
getOuterNodes() : List<CNode>	60	7
getDistancesPointToLine(List<Edge>) : double[]	59	5
getDistancesCornerToPoint() : double[]	57	6
GetPointsInsidePerpendicularPlanes() : List<Edge>	55	4
getPointsInsidePerpendicularPlane(double, double, double, doub	44	11

Obrázek 3: Naměřené hodnoty metrik třídy Calculate

Na obrázku 5 můžete vidět hodnoty cyklomatické komplexity a indexu udržitelnosti mé funkce. Nejvyšší komplexita je u poslední funkce 11, toto číslo se dá ale pořád považovat za dostatečně nízké. Z výsledků lze vyčíst, že všechny mé funkce jsou dostatečně jednoduché a kód je podle této metriky dobře testovatelný, jednoduchý a snadno pochopitelný.

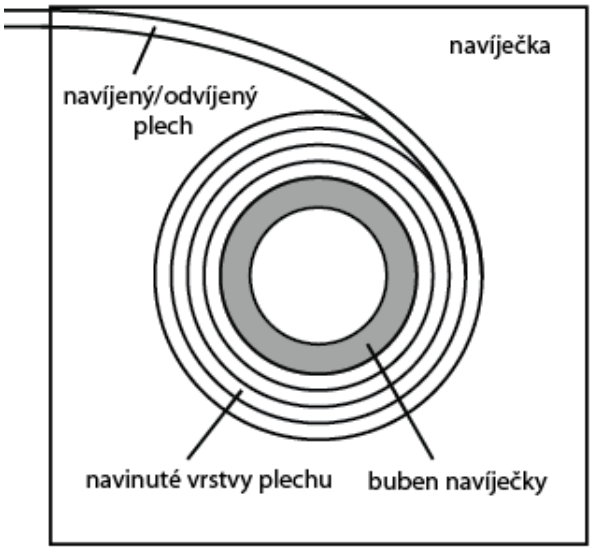
3.3 Funkce a testovací GUI pro výpočet teplot plechu na navíječce

Druhá časově náročnější práce spočívala v napsání funkce pro výpočet teplot chladnoucího plechu navinutého na navíječce po vyjetí z válcovací stolice.

3.3.1 Problém

Jednou z možností tváření kovů je válcování kovů. K tomuto účelu se používají tzv. válcovací stolice, kde se průchodem materiálu mezi otáčejícími se válci materiál deformuje do požadovaného tvaru profilu.

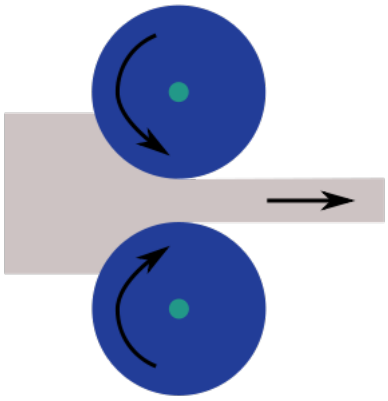
Má funkce byla vytvořena pro potřeby výpočtového modelování teplot pásu při jeho válcování na reverzní válcovací stolici s navíječkami.



Obrázek 4: Schéma navíječky

Obrázek 5: Ukázky profilů po průchodu válcovací stolicí. Obrázek převzat z internetu [4]

Na vstupu do reverzní válcovací stolice k válcování plechu (dále jen stolice) je plech kvůli své velké délce (v řádech stovkách metrů) z praktických důvodů navinutý na tzv. navíječce. Průchodem stolice se tloušťka plechu zmenší, na druhé straně se plech opět navíjí na navíječku. Většinou však nelze jedním průchodem plechu stolicí dosáhnout požadované tloušťky napoprvé a je nutný vícenásobný průchod. Navinutý plech je po dobu nutnou k přenastavení stolice na válcování v druhém směru chvíli svinut na navíječce, kde chladne nebo se ohřívá, podle typu navíječky, a poté je opět odvinut.



Obrázek 6: Princip zmenšování tloušťky plechů mezi válci válcovací stolice. Obrázek převzat z internetu [4]

3.3.2 Zadání

Mým úkolem bylo vypočítat teploty v referenčních bodech (dále jen RP) danými souřadnicemi po délce pásu, které jsou zadanou rychlostí navinuty na navíječku, určitý čas na navíječce chladne a pak je zase jinou rychlostí odvinut od pásu. Dodatečně zadání bylo později rozšířeno o část pásu, který zůstane nenavinutý mimo navíječku, pro který taktéž bylo třeba vypočítat teploty RP. Napsaná funkce se měla stát součástí velké výpočetní knihovny `TempFEM1D`.

Pro samotný teplotní výpočet MKP sítí v profilech nenavinutého i navinutého pásu na navíječce jsem měl k dispozici funkci `T_Calculate` ze zmíněné knihovny `TempFEM1D`.

3.3.3 Vstupní data

Na vstupu do funkce se předávaly čtyři vstupní parametry celkově třech typů struktur.

- `MATERIAL_DATA` - obsahovala údaje o vlastnostech materiálu pásu
- `COOL_SECTION` pro cívku na navíječce - obecnější struktůra, která obsahovala informace o emisivitě v jádru a v okolí bubnu navíječky apod.

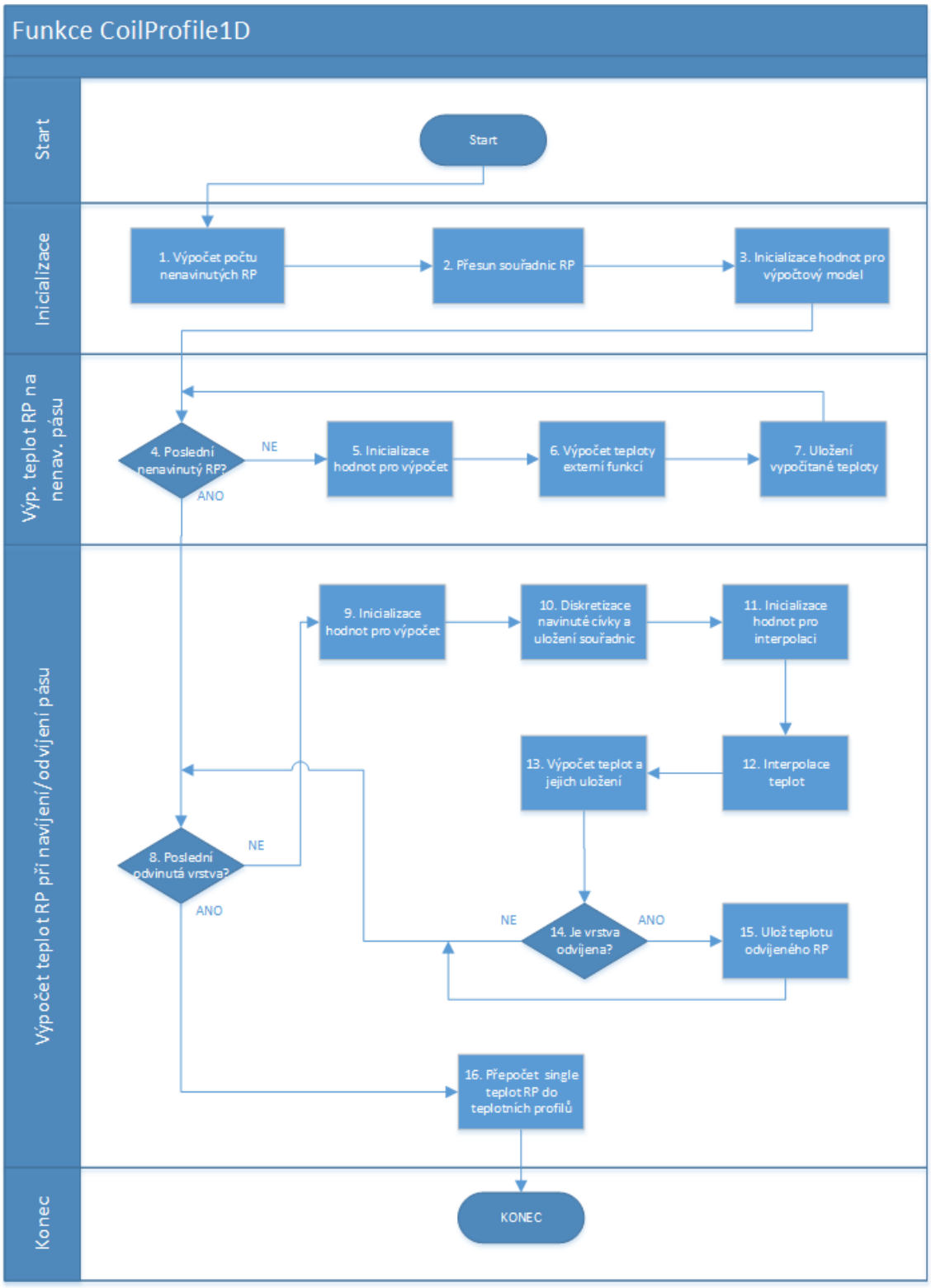
- `COOL_SECTION` pro nenavinutý pás - ta samá struktůra jako pro cívku na navíječce, pouze s hodnotami pro nenavinutý pás, obsahovala tedy informace o emisivitě na povrchu a pod povrchem pásu apod.
- `COILING_DATA` - struktůra vzniklá z dodatečných potřeb při vývoji funkce, v niž se předávají dodatečné parametry, které neobsahovaly výše zmíněné struktůry jako je délka pásu, počet RP na pásu, časy navíjení/odvíjení jednotlivých RP, délka reverzace a především nejdůležitější vstupně-výstupní pole s teplotama v RP

3.3.4 Požadovaný výstup

Primárním požadovaným výstupem funkce bylo přepočítané pole teplot v RP na výstupu z navíječky. Pro ověření správnosti algoritmu ale bylo nutné předat na výstup i průběh změn teplot v každém časovém kroku po navinutí či odvinutí jednoho RP a pole souřadnic navinutých RP na bubnu navíječky.

3.3.5 Řešení

V následujícím vývojovém diagramu funkce 7 je zjednodušeně zobrazen celý průběh funkce.



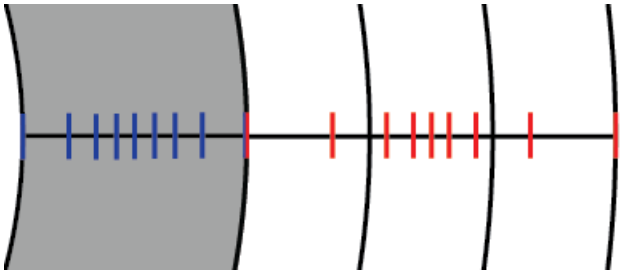
Obrázek 7: Vývojový diagram funkce CoilProfile1D

Pro vysvětlení algoritmu se nyní budu odkazovat k jednotlivým procesům a rozhodováním, které jsem pro tento účel v diagramu očísloval.

Na začátku algoritmu se v procesu `1` ze vstupního parametru délky nenavinutého pásu a z pole, které obsahuje rozestupy jednotlivých RP mezi sebou zjistí, kolik RP zůstane nenavinutých mimo navíječku.

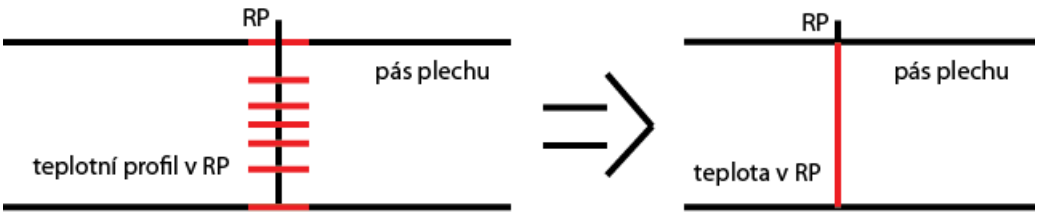
V procesu `2` se kontroluje zda-li nevychází konec délky nenavinutého pásu do polohy nějakého RP na pásu. Není-li konec nenavinutého pásu v poloze nějakého RP, přesune se poloha nejbližšího RP do tohoto místa, jinak se přesun neprovede. To z důvodu co nejmenšího zkreslení výstupních teplot funkce.

V procesu `3` se následně inicializují všechny potřebné pole, proměnné a struktůry. Zde se diskretizuje vnitřní síť jádra bubnu navíječky, jelikož tato síť zůstava po celou dobu výpočtu neměnná. Diskretizací se myslí rozvrstvení celé tloušťky materiálu tělesa do diskrétních uzlů, ve kterých se následně počítají jednotlivé teploty tzv. Metodou konečných prvků.



Obrázek 8: Detail diskretizace bubnu a navíječky při 3 navinutých vrstvách

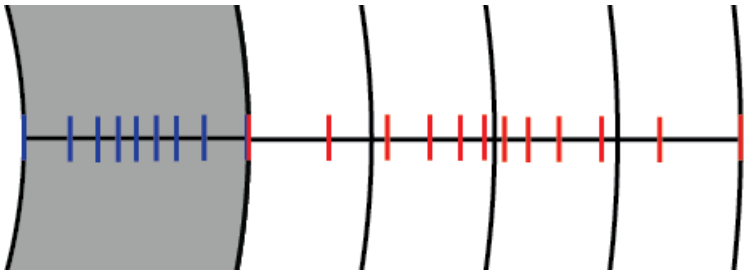
Na vstupu i výstupu funkce nejsou jednotlivé teploty v RP ale tzv. teplotní profily v RP. Pro zjednodušení algoritmu se každý teplotní profil RP na pásu (pouze na pásu, který bude navinut na buben navíječky a nezůstane nenavinut mimo navíječku) před hlavním výpočtem přepočítá do jediné průměrné teploty. Nyní je každému RP přiřazena pouze jedna teplota.



Obrázek 9: Princip přepočtu teplotního profilu do jediné teploty

`4–7` - Vypočítají se teplotní profily v RP, které nebudou navinuty na navíječku. Výpočet teplot provádí funkce `T_Calculate` z externí knihovny. Funkce se zavolá pro každý RP a výstupem funkce jsou již přímo teplotní profily v RP, které jsou požadovany na výstupu funkce.

`8-15` - V dalším hlavním cyklu se počítají teploty ve zbylých RP, tedy v těch RP, které budou navinuty na buben navíječky. Po inicializaci všech potřebných hodnot v `9` jako je výpočet počtu diskrétních uzlů po tloušťce cívky nebo výpočet tloušťky cívky s navinutým plechem se přejde k diskretizaci navinuté cívky v `10`. Diskretizace probíhá v každém cyklu znovu, protože je nutné s nově navinutou vrstvou plechu na jádru navíječky a s rostoucí šířkou cívky přidat počet diskrétních uzlů a zdiskretizovat nově přidanou vrstvu, či v případě odvíjení počet diskrétních uzlů naopak snížit a síť uzlů opět přepočítat. Jak můžete vidět na obrázku 8 v porovnání s obrázkem 10 je s navinutou novou vrstvou červená síť uzlů v navinuté cívce zcela odlišná. Modrá síť v bubnu však zůstava po celou dobu výpočtu neměnná.



Obrázek 10: Detail diskretizace bubnu a navíječky při 4 navinutých vrstvách

Jelikož síť uzlů je s každou navinutou/odvinout vrstvou zcela odlišná, je nutné v `11` a `12` pro výpočet přeinterpolovat teploty z uzlů předchozího cyklu do teplot v uzlech aktuálního cyklu.

Nyní je již možno v `13` opět zavolat funkci `T_Calculate`, která vypočte teploty v diskretizované síti. V `14` se kontroluje zda se nejedná o cyklus, ve kterém se vrstva odvíjí, je-li tomu tak, uloží se do dočasného pole teplota v posledním uzlu, jelikož tento uzel se nachází přesně na pozici odvíjeného RP. Po skončení hlavního cyklu po posledním odvinutém RP se v `16` překopírují teploty RP z dočasného pole, ve kterém jsou uloženy teploty při odvíjení RP, do požadovaných výstupních profilů v RP.

3.4 Grafický program

Z důvodů mé obliby k programovacímu jazyku C# jsem chtěl GUI nad algoritmem napsaným v C++ napsat právě v tomto jazyku. K této volbě mě taky táhlo to, že v již samotném .NET frameworku, který nad jazykem C# pracuje, jsou dostupné vhodné komponenty pro zobrazování grafů,, s kterými je velice jednoduché pracovat a s kterými jsem již měl navíc i zkušenosti. Nevýhodou tohoto řešení však byla nejen nutnost volat ze C# funkci napsanou v C++, ale především vyřešit předávání vstupních a výstupních dat z funkce.

V GUI byly požadovány tyto funkce:

- možnost nastavení všech vstupních parametrů funkce

- zobrazení průběhu teplot navinutého pásu na bubně navíječky po každém navinutí/odvinutí jednoho RP formou grafu
- výpis teplot a souřadnic navinutých RP v každém časovém kroku do tabulky
- zobrazení grafu teplot RP po opuštění navíječky

3.5 Volání funkce a předávání dat z C++ do C#

Původně jsem uvažoval o předávání dat pouze pomocí nějakého textového či binárního souboru, kde by se ukládaly a načítaly data předem daným formátem. Tato možnost však byla nevhodná z důvodu nutnosti předávání relativně velkého objemu dat a tak se od této možnosti opustilo. Zbývalo tedy jediné možné řešení a to předávání dat přímo mezi C++ a C# kódem.

Volání C++ funkcí z C# kódu je naštěstí z historických důvodů v dnešní době vcelku častá věc. Vývojáři často naráží na problém, kdy potřebují používat nějaké starší knihovny napsané v kdysi nejpoužívanějších programovacích jazycích a to C/C++. Microsoft, tvůrce jazyku C# a .NET frameworku, si tuto potřebu uvědomoval a vytvořil nad jádrem .NET frameworku tzv. Platform Invocation Services zkráceně P/Invoke který tento problém řeší.

P/Inovke je způsob volání nativního neboli unmanaged kódu (tedy kódu přeloženého do strojového kódu CPU) z tzv. managed kódu - kódu napsaného v jednom z .NET programovacích jazyků - C# nebo Visual Basic. Dále se pokusím na příkladu vysvětlit jeho princip.

Jako první bylo nutné překompilovat knihovnu z formátu .lib do .dll formátu. Dosáhnout tohoto je velice jednoduché, stačí pouze ve Visual Studiu 2012 v nastavení projektu knihovny změnit konfigurační typ ze "Static library (.lib)"na "Dynamic library (.dll)".

Následně je potřeba nějakým způsobem říct kompilátoru jaké má externí unmanaged funkce vstupní parametry a návratovou hodnotu neboli jak je funkce deklarována. Pro tento účel je nutné vytvořit novou statickou třídu, která obsahuje deklaraci funkce srozumitelnou C# kompilátoru.

V mém případě deklarace funkce v C++ vypadá jako ve výpisu kódu 2.

```
int Coil_Profile1D(
    COOL_SECTION *pCoolCoil,
    COOL_SECTION *pCoolStrip,
    MATERIAL_DATA *pMaterialData,
    COILING_DATA *pCoilingData,
    double **&retTempsOfMeshesInAllTimes,
    double **&retCoordsOfMeshesInAllTimes,
    int &retMaxMeshElementCount,
    int &retRowsCount
    );
```

Výpis 2: Deklarace funkce v C++

Tato deklarace byla přepsána do C# jako ve výpisu kódu 3.

```
[DllImport("CoilProfile.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int Coil_Profile1D(
    MCOOL_SECTION pCoolCoil,
    MCOOL_SECTION pCoolStrip,
    MMATERIAL_DATA pMaterialData,
    MCOILING_DATA pCoilingData,
    ref IntPtr retTempsOfMeshesInAllTimes,
    ref IntPtr retCoordsOfMeshesInAllTimes,
    ref IntPtr retMeshElementCount,
    ref IntPtr retRowsCount
);
```

Výpis 3: Deklarace funkce v C#

Před deklarací funkce je nutné použit attribut DllImport, který říká překladači v jakém .dll souboru se funkce nachází a jak se má funkce volat.

Dále jste si mohli všimnout, že struktůry v C# se jmenují jinak než v C++ a to ze stejného důvodu jako u funkce, protože v tuto chvíli by překladač neznal deklaraci struktůry stejně jako ji neznal u funkce. Je teda nutné podobným způsobem jako při deklaraci funkce přepsat i deklarace struktur do C#.

Jako příklad deklarace struktůry `MATERIAL_DATA` v C++ vypadá jako ve výpisu kódu 4.

```
typedef struct {
    double dC_X[DIM];
    double dC_Y[DIM];
    double dRO_X[DIM];
    double dRO_Y[DIM];
    double dLAM_X[DIM];
    double dLAM_Y[DIM];
    int iCount_C;
    double dA;
    double dKoef[10];
} MATERIAL_DATA;
```

Výpis 4: Deklarace struktury v C++

Tato struktůra z 4 byla přepsána do C# jako ve výpisu kódu 5.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct MMATERIAL_DATA{
    public IntPtr dC_X;
    public IntPtr dC_Y;
    public IntPtr dRO_X;
    public IntPtr dRO_Y;
    public IntPtr dLAM_X;
    public IntPtr dLAM_Y;
    public int iCount_C;
    public double dA;
    public IntPtr dKoef;
};
```

Výpis 5: Deklarace struktury v C#

Opět je zde použit jeden atribut - StructLayout který informuje překladač o způsobu uložení struktůry v paměti a její znakové sadě. U všech polí bez ohledu na jejich datový typ se používá datový typ IntPtr, který je chápan jako obecný ukazatel do paměti. Pojmenování atributů nemá zcela žádný vliv na funkci, je však nutné zachovat pořadí.

Jakmile máme tyto úpravy hotové, je možné po naplnění struktur a jejich předání funkci jednoduše zavolat.

```
CoilProfile .Coil_Profile1D(
    pCoolCoil,
    pCoolStrip,
    pMaterialData,
    pCoilingData,
    ref  retTempsOfMeshesInAllTimes,
    ref  retCoordsOfMeshesInAllTimes,
    ref  retMaxMeshElementCount,
    ref  retRowsCount
);
```

Výpis 6: Volání funkce ze C#

Poslední čtyři parametry funkce byly vytvořeny dodatečně pouze pro účely testovacího GUI. Parametr retTempsOfMeshesInAllTimes je ukazatel na výstupní pole, ve kterém se nachází teploty všech RP po každém navinutí/odvinutí jednoho RP. V poli ukazatele retCoordsOfMeshesInAllTimes jsou uloženy souřadnice jednotlivých RP navinutých na cívce v každém časovém kroku. Poslední dva parametry slouží k transformaci jednorozměrných polí retTempsOfMeshesInAllTimes a retCoordsOfMeshesInAllTimes do dvourozměrných. Tyto pole jsou v těle funkce dvourozměrné, avšak pro snažší předávání mezi C++ a C# kódem je vhodné překopírovat si tyto pole do jednorozměrných a v C# kódu zpět do dvourozměrných polí. Parametr retMaxMeshElementCount určuje dimenzi X, retRowsCount dimenzi Y obou polí zároveň.

Po zavolání funkce, jelikož předané parametry retTempsOfMeshesInAllTimes a retCoordsOfMeshesInAllTimes jsou pouze ukazatelé do paměti, je nutné si vytvoři nové lokální pole se stejnou velikostí jako pole na ukazatelích a z těchto ukazatelů si obsah polí překopírovat.

```
double[] temps = new double[retMaxMeshElementCount*retRowsCount];
Marshal.Copy(retTempsOfMeshesInAllTimes, temps, 0, retMaxMeshElementCount*retRowsCount);
```

Výpis 7: Získání výstupních dat

Nyní již v poli temps máme všechny hodnoty z výstupního pole funkce a s polem může být libovolně zacházeno. V mém případě byly hodnoty nahrány do kolekce List jenž obsahovala instance mnou vytvořené třídy MKP a ty byly následně zobrazené v grafech programu.

3.6 Časová náročnost úkolů

První úkol mi zabral v porovnání s druhým méně času. To bylo zapříčeněno jasnou definicí vstupů a požadovaného výstupu které se narozdíl od druhého úkolu v průběhu práce neměnil. Celková doba strávená na prvním úkolu byla 20 dní.

Druhý úkol byl časově náročnější. To bylo způsobeno nutností pochopení funkčnosti používaných metod z již napsané knihovny, pochopení významu struktur, spousty změn a dodatečných úprav v průběhu práce na algoritmu a taky dlouhého ladění algoritmu a nadstavbového GUI, které bylo napsané v odlišném programovacím jazyku. Celková doba strávená na druhém úkolu byla 30 dní.

4 Teoretické a praktické znalosti získané v průběhu studia a uplatněné v průběhu praxe

Při práci na prvním i druhém úkolu jsem zužitkoval znalosti z programování v C++ z předmětu Základy programování. V programování v jazyce C# jsem využil znalosti z předmětů Programovací jazyky II a Architektura technologie .NET. Pro potřeby počítání rovnic přímek a kolmic, hledání bodů ležících ve vymezené ploše v mé první práci jsem zužitkoval nabyté znalosti z předmětu Matematické analýzy pro IT. Při tvorbě GUI v druhém úkolu jsem uplatnil znalosti z předmětu Uživatelská rozhranní.

5 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Jako jediný problém, na který jsem narazil a musel jsem se danou problematiku doučit, bylo v případě volání C++ kódu z jazyku C# a následného předání dat mezi těmito jazyky metodou P/Invoke.

6 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Zvolení praxe místo klasické bakalářské práce nelituji a zvolil bych ji znovu. Nejvíce pozitivně hodnotím nabytou zkušenost práce ve skutečné firmě na skutečných problémech, kterou jsem předtím neměl. Také jsem si uvědomil jak je důležitá komunikace při práci a detailní popis zadání, který snižuje počet možných chyb v kódu. Navíc jsem nahlédl do oboru o kterém jsem měl jen málo tušení a to teplotní simulace, metalurgie a hutnictví s jimiž se nejspíše už dále v práci nepotkám.

7 Reference

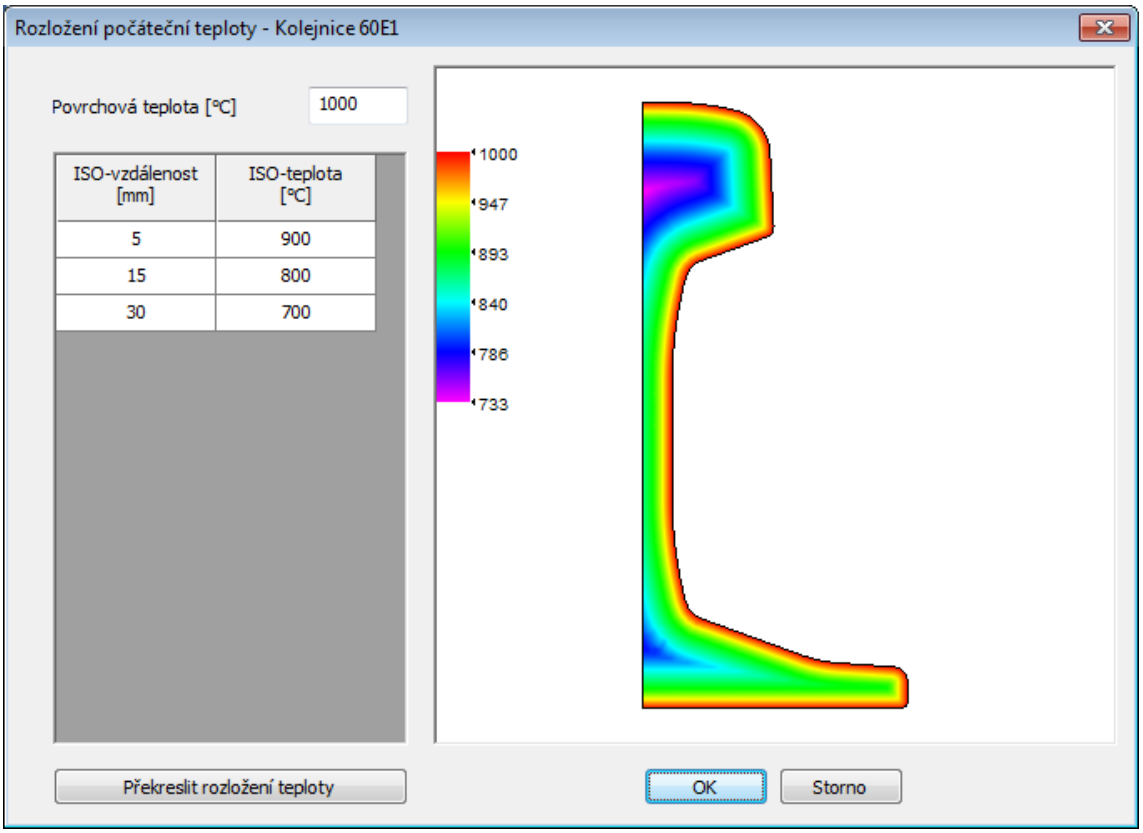
[1] Společnost ITA. [online]. [cit. 2014-04-27]. Dostupné z: http://www.ita-tech.cz/cs/hlavni-stranka

[2] Počítačové simulace MKP [online]. [cit. 2014-04-27]. Dostupné z: http://www.ita-tech.cz/cs/produkty-a-sluzby/pocitacove-simulace-mkp/

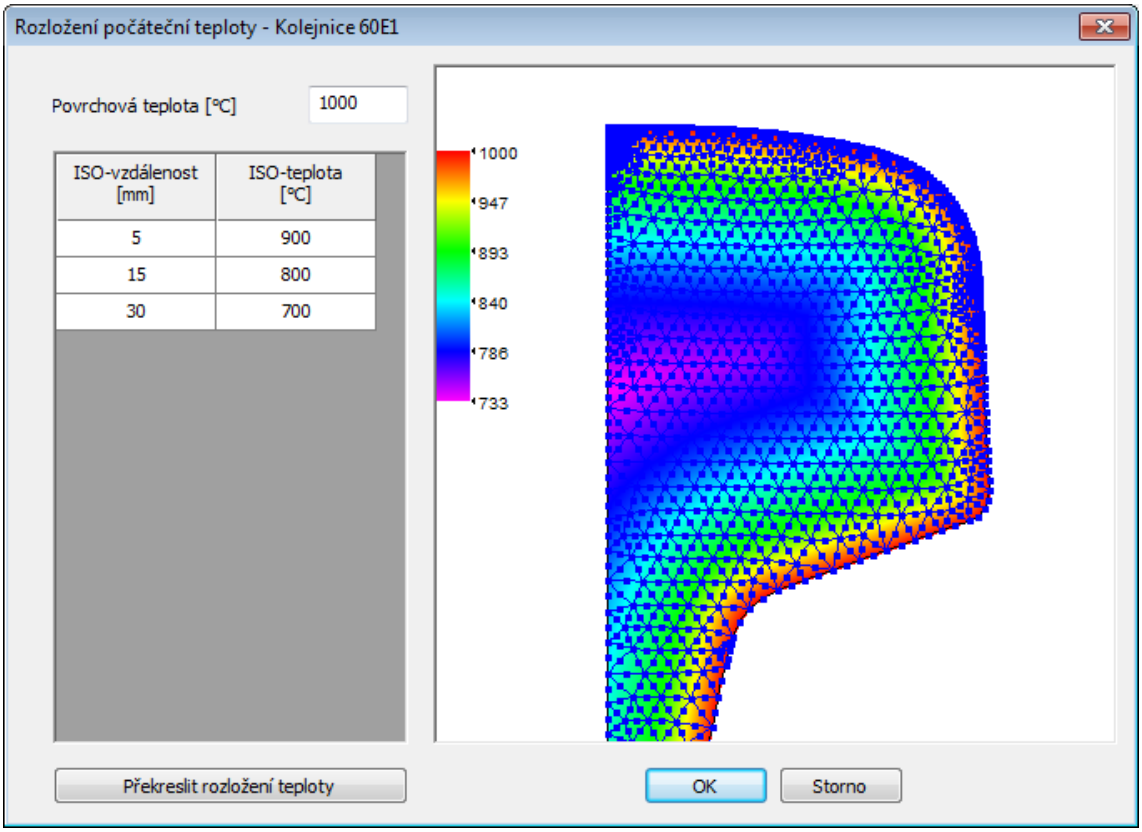
[3] Cyklomatická složitost - Wikipedia. [online]. [cit. 2014-04-27]. Dostupné z: http://cs.wikipedia.org/wiki/Cyklomatická-složitost

[4] Rolling (metalworking) - Wikipedia. [online]. [cit. 2014-04-27]. http://en.wikipedia.org/wiki/Rolling-(metalworking)

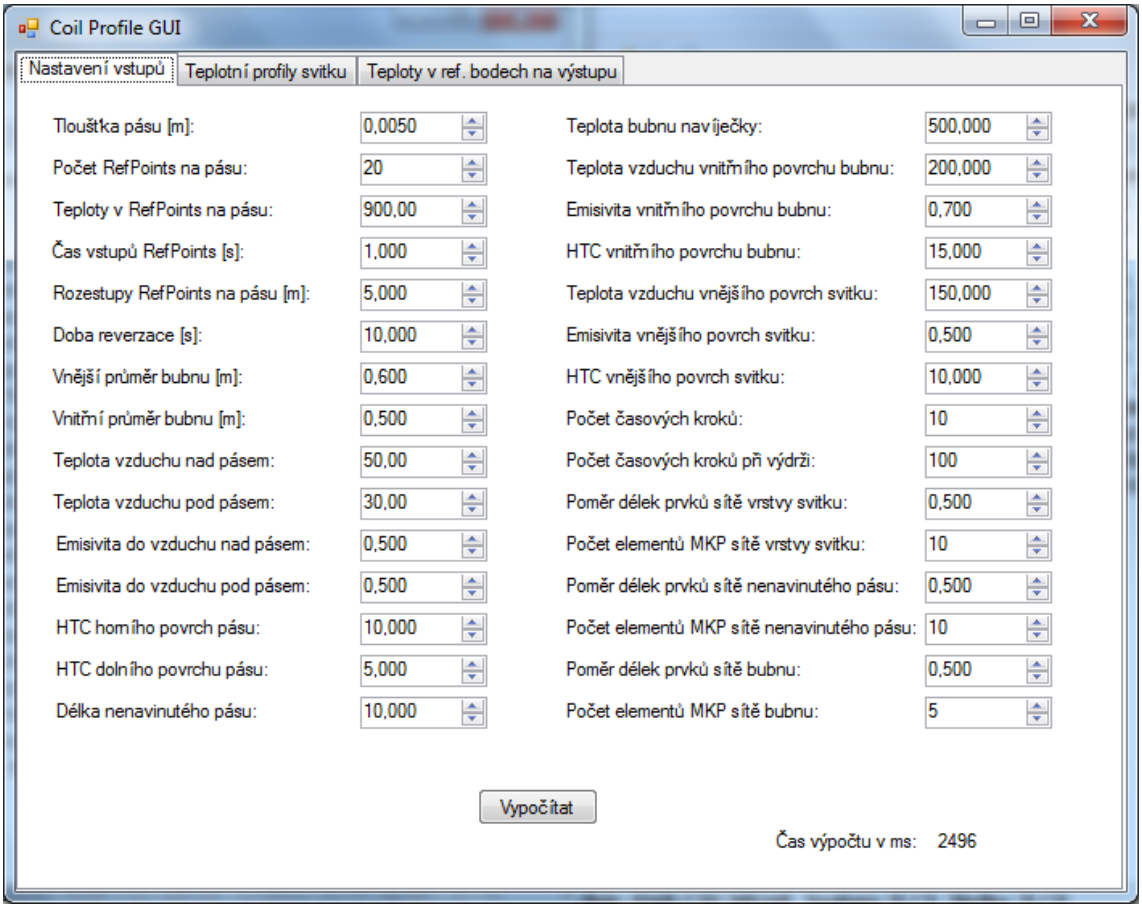
A Přílohy



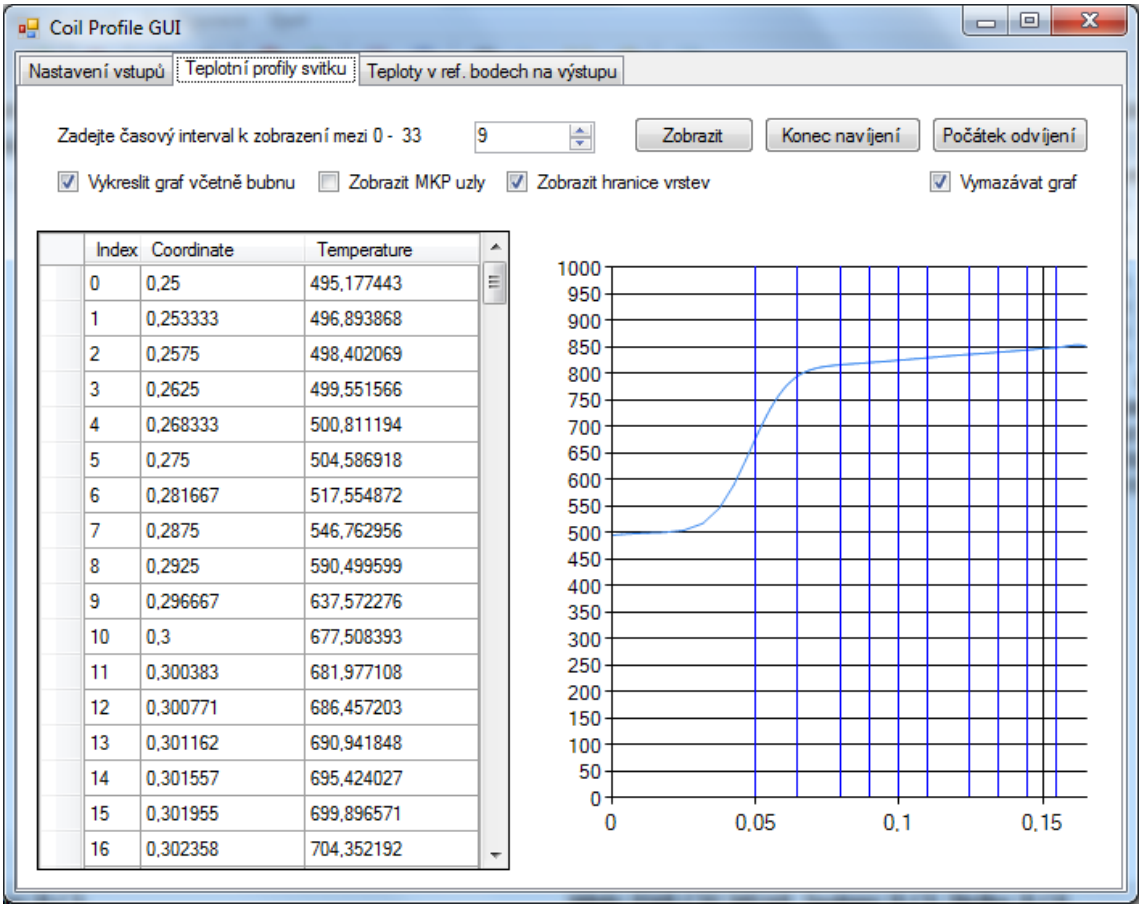
Obrázek 11: Program 1 - vykreslení počáteční teploty po průřezu kolejnice podle teplot aproximovaných do uzlů MKP sítě



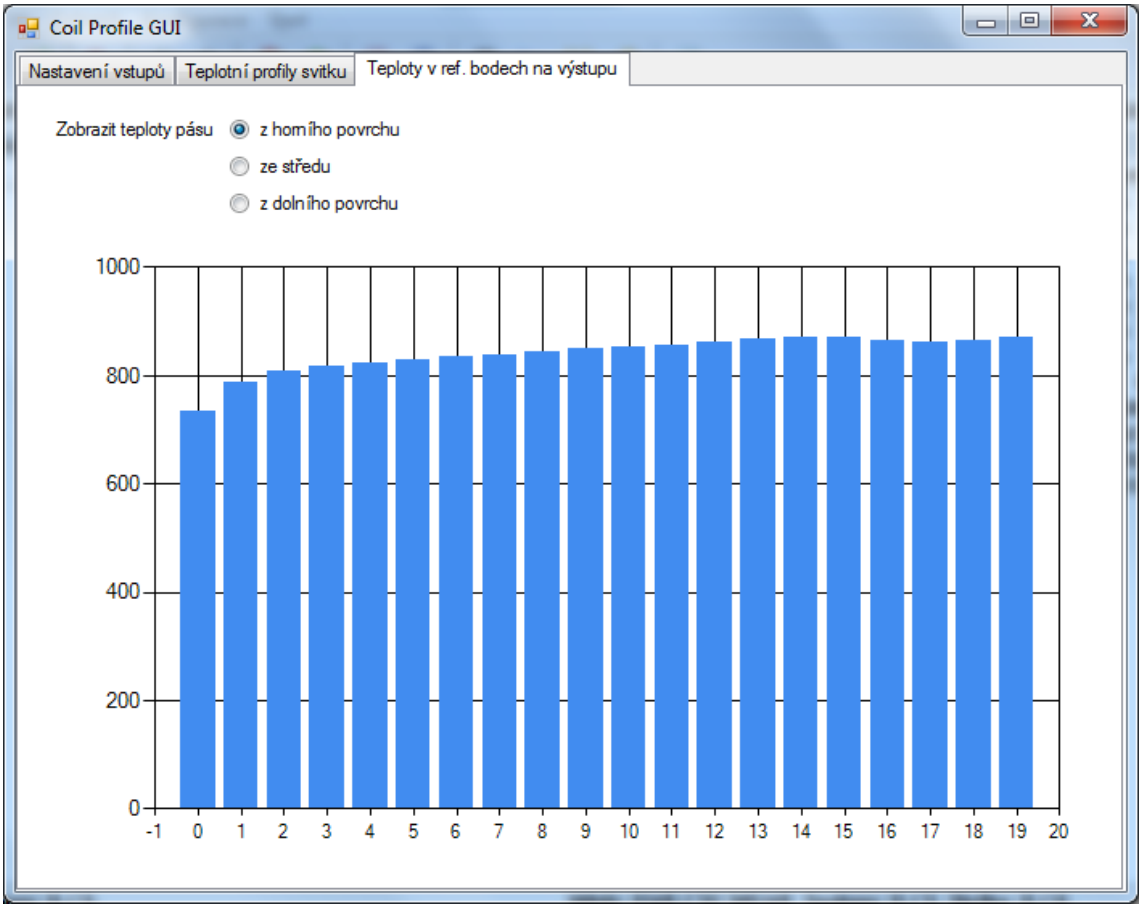
Obrázek 12: Program 1 - detail MKP sítě po průřezu tělesa



Obrázek 13: Program 2 - okno s nastavením vstupních parametrů



Obrázek 14: Program 2 - zobrazení průběhu teploty po průřezu svitkem ve vybraném časovém kroku při jeho navíjení (vlevo vnitřní povrch trnu, vpravo vnější povrch svitku)



Obrázek 15: Program 2 - zobrazení teplot v RP pásu na výstupu z navíječky